Lab 5: Using Venus

Taaruni Ananya | Session 008 | CRN: 15623

02/21/2025

Dr. Lan Gao

Introduction

Lab 5: Using Venus is centered around building on topics introduced and explored from Lab 4: Introduction to NASM, along with add and call. Lab 5 looks into learning how to use tools such as VS Code and Terminal, extensions such as riscv-venus and RISC-V Support, and Venus. The lab explains how comments and commands work, breaks down code presented in this Lab's file, and shows C code to explain deeper. The lab also gives hands-on experience with running and debugging menus in VS Code.

Purpose

The purpose of this lab is to continue practicing the use of Assembly code, along with the terminal and commands; this lab also works to introduce Venus, extensions, and using run and debug features on VS Code to step through a program. Lab 5 is mainly focused on introducing Venus and features of VS Code. Lab 5 will also delve into teaching “run”, “debug”, “continue”, “restart”, “step out”, “step over”, and “step into” commands, how code is stored in a text section, comments and using comments, and the different commands and their respective outcomes.

Main Objectives

- Learn and use VS Code
- Learn “run”, “debug”, “continue”, “restart”, “step out”, “step over”, and “step into” commands
- Learn assembly code, environment call, loops, and how data is stored.

Brief Explanation

VS Code is an integrated development environment (IDE) where Venus can be accessed and programmed. The HelloWorld.S file is an Assembly code that prints the message “Hello RISC-V World” followed by “42” and the number of iterations it is at. This lab goes through the program as it loops, keeps track, stores memory, and prints these outputs. Instead of executing the infinite

loop, the “step into” feature on VS Code through “run and debug” makes it possible to step through the program and print as many times as necessary.

Methodology/Procedure

The procedure starts with introducing the code and how comments work with ‘#’. The instructions then dive deeper into explaining the commands and what each line of code does to build the program. Then, the “run and debug” feature is used–specifically “step”--to go through the program and print the output three times to test.

Documentation

Initializes the counter:

```
mv    s0, x0
```

Increments the counter:

```
loop:   addi  s0, s0, 1
```

Jumps to the loop again

```
j       loop
```

Output

```
Hello RISC-V World
42 1
Hello RISC-V World
42 2
Hello RISC-V World
42 3

Stop program execution!
-------------------------------------------------------------------------------
```

HelloWorld.S code:

```
.text
        mv    s0, x0
loop:   addi  s0, s0, 1     # s0 holds the count

        # Print the helloworld string
        li    a0, 4           # print a string
        la    a1, helloworld  # load address of helloworld string
        ecall

        # print the number 42
        li    a0, 1      # print an integer
        li    a1, 42     # Why 42? See The Hitchiker's Guide to the Galaxy, Chapter 27
        ecall

        li    a0, 11     # print a character
        li    a1, 32     # space (the character to print)
        ecall

        li    a0, 1      # print an integer
        mv    a1, s0     # s0 contains the count
        ecall

        li    a0, 11     # print a character
        li    a1, 10     # newline
        ecall
```

```
        li    a0, 11     # print a character
        li    a1, 10     # newline
        ecall

        j     loop
.data
helloworld:      .string "Hello RISC-V World\n"
```

Questions:

Part 1:

1. **Suppose that we have the following code:**

   li a0, 1

   li a1, 100

   ecall

   What do you expect will happen when this is executed?

   The first line li a0, 1 is a system call that prints an integer–indicated by the ‘1’. The second line .li a1, 100 assigns 100 to a1. Finally, the third line ecall instruction calls to print the integer, which is 100 in this case.

2. **Suppose that we have the following code:**

   li a0, 11

   li a1, 100

   ecall

   What do you expect will happen when this is executed?

   The first line li a0, 1 is a system call that prints a character–indicated by the ‘11’. The second line .li a1, 100 assigns 100 to a1. Finally, the third line ecall instruction calls to print the character, which is 100 in this case.

Part 2:

1. **You should notice how the "PC" value on the left (under Variables) changes as you step through the program. What does PC stand for in this context? Why does it change from 0x00000004 to 0x00000008 in one step?:**

According to Lenovo, PC stands for program counter in this context; it is “a fundamental component of a computer's central processing unit (CPU). It is a special register that keeps track of the memory address of the next instruction to be executed in a program”. It changes from 0x00000004 to 0x00000008 in one step because the RISC-V instruction is 4 bytes and increments by 4–leading the change in one step.

2. **Examine the "Integer" values on the left (under Variables). Describe what happens with x10 and x11 as you step through the code. Give an example of one line of the code, and what effect it has on x10. Give an example of one line of the code, and what effect it has on x11.**

   x10 as a0 in `li a0, 4` is set to 4, changing x10 from 0x00000000 to 0x00000004. x11 as a1 on the other hand as `li a1, 42` loads 42 into x11, changing the value just like it did for x10 with storing the value 42.

3. **What does the "Continue" button do? Describe it in your own words.**

   The “Continue” button runs through the program completely without braking until it finishes its execution.

4. **What does the "Step-Out" button do? Describe it in your own words.**

The “Step-Out” button finishes the execution where it is and goes back to the code’s beingging. For example, if this button is activated in the middle of executing a block of code, it will complete and revert back to where it was called.

5. **What is the difference between "Step Over" and "Step Into"? Describe it in your own words.**

   The “Step Over” button skips a function that’s ahead of where it is right now. For example, if it’s done executing one function’s code block, it will skip over the next function without going into it line by line and continue after the latter. The “Step Into” button on the other hand, enters the function and its each line of code to execute. For example, it will enter a function and execute line by line, showing what each line does.

Key Code Observations

```
li    a0, 4
```

```
li    a1, 42
```

```
ecall
```

```
mv    s0, x0
addi  s0, s0, 1
```

```
j     loop
```